Autonomous Vehicle Control: End-to-end Learning in Simulated Environments

Hege Haavaldsen* Max Aasbø* Håkon Hukkelås
Frank Lindseth †

Abstract

This paper examines end-to-end learning for autonomous vehicles in diverse, simulated environments containing other vehicles, traffic lights, and traffic signs; in weather conditions ranging from sunny to heavy rain. The paper proposes an architecture combing a traditional Convolutional Neural Network with a recurrent layer to facilitate the learning of both spatial and temporal relationships. Furthermore, the paper suggests a model that supports navigational input from the user to facilitate the use of a global route planner to achieve a more comprehensive system.

The paper also explores some of the uncertainties regarding the implementation of end-to-end systems. Specifically, how a system's overall performance is affected by the size of the training dataset, the allowed prediction frequency, and the number of hidden states in the system's recurrent module. The proposed system is trained using expert driving data captured in various simulated settings and evaluated by its real-time driving performance in unseen simulated environments.

The results of the paper indicate that end-to-end systems can operate autonomously in simulated environments, in a range of different weather conditions. Additionally, it was found that using ten hidden states for the system's recurrent module was optimal. The results further show that the system was sensitive to small reductions in dataset size and that a prediction frequency of 15 Hz was required for the system to perform at its full potential.

1 Introduction

In recent years, substantial progress has been made towards a vehicle's ability to operate autonomously. Primarily, two different approaches have emerged; module-based and end-to-end based methods. The prevailing state of the art approach is to divide the problem into a number of sub-problems and solve them by combining techniques like mapping and localization, perception and prediction, and path planning and control. This approach requires expert knowledge in several domains and often results in complex solutions, consisting of several cooperating modules.

On the other side of the spectrum, we have end-to-end approaches. End-to-end models are designed to utilize minimal expert domain knowledge and instead relies on machine

*These authors contributed equally to this work.

†Corresponding author.

This paper was presented at the NIK-2019 conference; see http://www.nik.no/.

learning to learn a mapping from the perceived environment into a sequence of actions. End-to-end models are, by design, black box methods, making them hard to explain and untrustworthy. However, with the rapid advances in deep learning and the increasing abundance of open-source datasets, end-to-end solutions are showing great potential.

Traditionally, end-to-end models are trained in a supervised manner, studying data from an expert driver, and trying to imitate the experts' actions. Even though the end-to-end model is able to perceive the current environment, it is not able to make the correct navigational decisions solely based on its perception. Therefore, it is necessary to include a user's intent in situations that require a decision (e.g., when approaching an intersection). Hence, we desire that the end-to-end model is able to receive and adapt to navigational commands.

In this work, we investigate end-to-end models' ability to drive autonomously in complex simulated environments. We adapt the model from Haavaldsen *et al.* [1] to manage difficult environments, including complex driving scenarios with multiple agents in the environment, and challenging weather conditions. We present several ablation experiments to quantify the effect of specific design choices typically made when implementing an end-to-end system for autonomous vehicle control: including the number of frames to consider when predicting an action, the importance of the training set size, the number of hidden states in the LSTM module, and the value of prediction frequency. Finally, we perform extensive real-time testing on our best model. [1]

The rest of this paper is organized as follows. Section 2 presents related work, while 3 describes the problem definition of this paper. Section 4 addresses the environment in which the data were collected and where the experiments were conducted. Section 5 reviews the collection and preprocessing of the data. The model architectures are presented in Section 6. Section 7 and 8 covers the experimental setup and results, while section 9 discusses the results. Finally, Section 10 concludes the paper.

2 Related Work

Haavaldsen *et al.* [1] explores end-to-end systems capability to operate in simulated urban environments. Their model is based on the DAVE-2 architecture [2] and is trained in a supervised manner using expert driving data from simulated environments. Their empirical results indicate that the DAVE-2 architecture can significantly improve its autonomous driving capability by including temporal information, with the use of Long-Short-Term-Memory (LSTM) cells. However, their experiments are focused on simple urban environments with little to no variance in weather condition. In this work, we adapt their model. Specifically, we explore the models ability to act autonomously in more challenging environments; highways with several agents, complex urban environments, and a large variance in weather conditions. Furthermore, we perform several ablation experiments to improve upon their proposed model.

End-to-end learning for autonomous vehicles has become increasingly popular over the last decades. Pomerleau *et al.* [3] proposed the first end-to-end model back in 1989, which was a fully-connected neural network. Later on, the first promising proof-of-concept project emerged in 2003, known as DAVE [4]. DAVE was able to fully autonomously drive around a junk-filled alley while avoiding obstacles. Three years later NVIDIA developed DAVE-2 [2], a framework with the objective to make real vehicles drive reliably on public roads. DAVE-2 is the basis for most end-to-end approaches seen

[1] Examples of our model driving can be seen here: `www.tinyurl.com/e2e-example`

today [5, 6]. The project used a CNN to predict a vehicle's steering commands. Their model was able to operate on roads with or without lane markings and other vehicles, as well as parking lots and unpaved roads.

Codevilla *et al.* [5] further explored NVIDIA's architecture by utilizing *Conditional Imitation Learning*, where navigational commands are given as input during training, allowing the model to function in scenarios that require decisions. This is expressed by Equation 1.

$$\underset{\theta}{\text{minimize}} \quad \sum_i L(F(o_i, c_i; \theta), a_i) \tag{1}$$

Here, F is a deep neural network, o_i is an observation, and c_i represents the user's intent. Given the input o_i and c_i, the model predicts an action $\hat{a}_i = F(o_i, c_i; \theta)$. The objective is to optimize a set of parameters θ, such that the difference between the estimated action, $\hat{a}_i$, and the expert's action, a_i, is minimized. By adding c_i, the model should be able to handle more scenarios than before and provide the ability for real-time control of the model. The authors proposed two network architectures: a *branched network* and a *command input network*. The *branched network* used the navigational input as a switch between a CNN and three fully connected networks, each specialized to a single intersection action, while the *command input network* concatenated the navigational command with the output of the CNN, connected to a single fully connected network.

Hubschneider *et al.* [6] proposed using turn signals as control commands to incorporate the steering commands into the network. Furthermore, they proposed a modified network architecture to improve driving accuracy. They used a CNN that receives an image and a turn indicator as input such that the model could be controlled in real-time. To handle sharp turns and obstacles along the road, the authors proposed using images recorded several meters back to obtain a spatial history of the environment. Images captured 4 and 8 meters behind the current position were added as an input to make up for the limited vision from a single centered camera.

Hesham *et al.* [7] proposed a model including temporal dependencies to predict an action. They combine a CNN with an LSTM network to utilize temporal dependencies. Additionally, the authors proposed formulating the steering angle prediction as a classification task, thus imposing a spatial relationship between the output layer neurons in the network. This was done by encoding the output as a sine wave and letting the steering angle correspond to the phase shift. This encoding is given in Equation 2.

$$Y_i = \sin\left(\frac{2\pi(i-1)}{N-1} - \frac{\phi\pi}{2\phi_{max}}\right), \quad 1 \leq i \leq N \tag{2}$$

Y_i is the activation of neuron i, and N is the number of classes of the output layer. They use $N = 95$ neurons and a max steering angle of $\phi_{max} = 190°$. Gradual changes in steering angle will then lead to gradual changes in the output layer's activations. , and their empirical results shows that a C-LSTM improves the angle and stability root mean square error by 35% and 87% on the comma.ai dataset [8] , respectively.

3 Problem Definition

This paper proposes a revised version of the conditional imitation learning from Codevilla *et al.* [5]. To account for both the user's intent and additional world information, the model takes an observation o_i, a user's intent h_i, and external state information s_i as

input in the decision making. h_i is a one-hot encoded vector representing the user's navigational intent, hereby referred to as the navigational input. The navigational input can either be: *Turn Left at Next Intersection*, *Turn Right at Next Intersection*, *Continue Straight at Next Intersection*, or *Follow Lane*. Moreover, additional information about the world, such as the current speed limit, the vehicle's speed, and the current traffic light state, are introduced by s_i, hereby referred to as the external state information. The revised imitation learning technique is expressed in Equation 3, where a model, F, learns a mapping between the inputs (i.e., o_i, h_i, s_i) and the executed action a_i, by fitting the learnable parameters, θ, to minimize the loss, L.

$$\underset{\theta}{\text{minimize}} \quad \sum_i L(F(o_i, h_i, s_i; \theta), a_i) \tag{3}$$

4 Environment

For this paper, the CARLA simulator [9] was used to gather training data and to evaluate the proposed model. CARLA is an open-source simulator built for autonomous driving research and provides an urban driving environment populated with buildings, vehicles, pedestrians, and intersections. The simulator allows customization and control of non-player agents and provides flexible setup of environments: such as type of sensors, number of vehicles, and weather conditions. CARLA comes with a range of different urban and suburban layouts out-of-the-box, differing in size and complexity.

By using a simulator, we could effectively set up a variety of corner cases needed for training, validation, and testing; while removing any safety risks and material costs. However, it is important to recognize that a simulation is only an imitation of a real-world system, and a model trained on only simulated data may not be able to function reliably in the real world. Nonetheless, a model's performance in a photo-realistic simulator serves as a good indication of its real-world performance and may help to illuminate weaknesses early on. Additionally, a simulator serves well for benchmarking different models as you are able to assure that any external factors are constant throughout the tests.

5 Data Generation

When performing imitation learning, the selection-process of training data plays a significant role in a model's ability to perform reliably in different conditions. We collect data from the CARLA simulator, following the collection technique proposed in Haavaldsen *et al.* [1].

Two different datasets were gathered, one from Town 1 and one from Town 4. All data were captured in environments without pedestrians, alongside a varying number of non-player vehicles. Some of the training data were captured entirely without any other vehicles, while some of the data were captured with a randomly generated amount (100-200) of other vehicles.

Data were gathered in seven different weather conditions, at noon and sunset. The different weather conditions consisted of clear sky, cloudy sky, soft rain, medium rain, hard rain, clear wet, and cloudy wet; resulting in a total of 14 different weather/lightning combinations.

The data was captured using a capture frequency of 10 observations each second. In total, 3.4 hours of training data were captured, 2.4 hours in Town 1, and 1.0 hour in Town 4. Table 1 summarizes the gathered datasets.

Town	No. of observations	Size	Duration
1	87 893	39.6 GB	2.4 h
4	33 757	14.1 GB	1.0 h

Table 1: The collected training data. An observation contains the captured data from a single rendered frame in the simulator. Data were captured with a frequency of 10 Hz.

Data augmentation is crucial for our models ability to generalize. Specifically, it is possible to simulate several of the varying environmental conditions using different image transformations. For each observation in the training set, we generate a new augmented sample by using one of the desirable transformations picked at random. These transformations includes: a random change in brightness, random changes in hue, the addition of Gaussian blur, the addition of simulating shadows, and the addition of simulated rain.

Another vital factor in end-to-end learning is the balance of target values within a dataset. A model trained on unbalanced data may incorrectly bias some actions over others. For example, when typically driving, a majority of the observations are the vehicle driving straight. To counteract this, we use a fraction of the examples with a small steering angle. Also, we increase the number of observations with a large steering angle by upsampling. Additionally, the observations corresponding to the different intersection decisions (i.e., turn left, turn right, or straight ahead) were balanced by analyzing the observations' *Navigational Input*-property and downsampling the over-represented choices. Finally, most of the observations where the vehicle was not moving (e.g., waiting for a red light) were downsampled.

6 Model Architecture

We propose a system consisting of two modules, a *Steer Predictor* and a *Throttle-Brake Predictor*. Each module takes a sequence of forward-facing images, navigational inputs, and external state information as input. The images are sent as input to a CNN, where each image is processed independently to extract important features from the images. The output from the CNN is sent to an LSTM alongside navigational inputs and external state information to also learn temporal dependencies between the subsequent images. The system overview is illustrated in Figure 1.

In the *Steer Predictor*-module, the output of the LSTM is passed through a fully connected layer, producing a steer angle prediction encoded as points forming a sine wave. Each of the neurons in the connected layer corresponds to a point on the predicted sine wave. To train the model, we compute the loss between the predicted steer sine wave and the encoded ground truth with the root mean square error (RMSE), and optimize the model with the adam optimizer [10]. In inference time, the steer prediction is decoded back into a steer angle by taking the phase shift of the predicted sine wave. Figure 2 illustrates the training and deployment phase of the steering angle prediction.

In the *Throttle-Brake Predictor*-module, the output of the LSTM is passed through a fully connected layer to predict a throttle and brake value. During training, the loss between the predicted values and the ground truth, are calculated using an MSE loss function, and optimized with the same optimizer as the steer predictor. During inference time, the output from the predictor yields the final throttle and brake predictions.

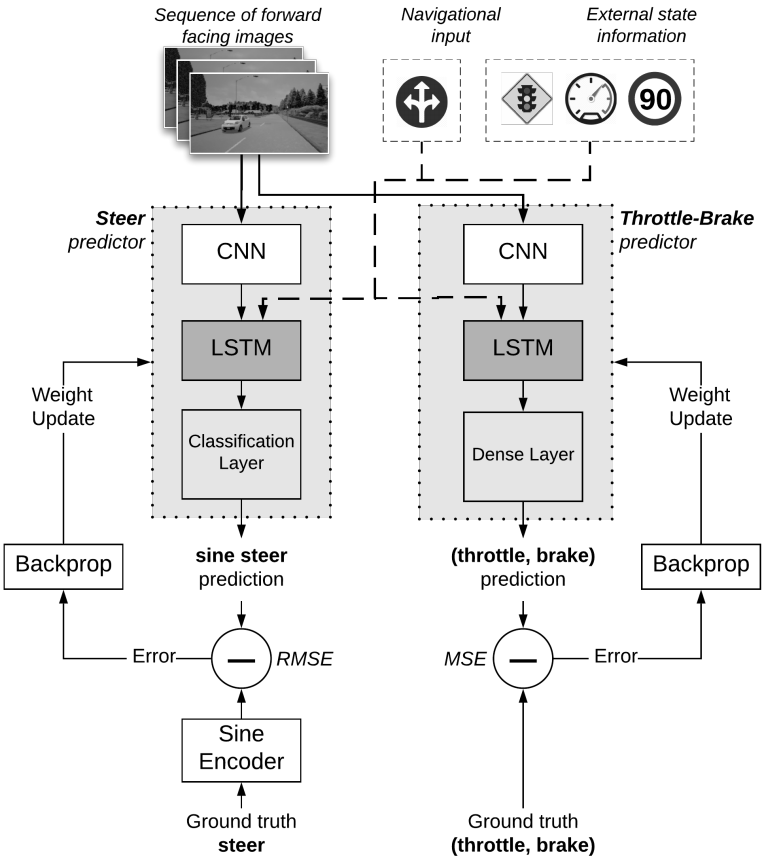

Figure 1: An overview of the propsed system in its training phase.

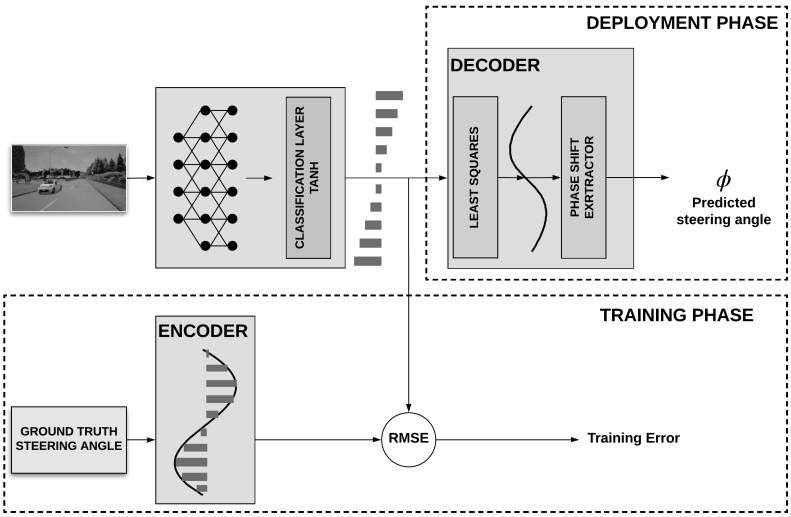

Figure 2: The training and deployment procedure of the steer angle prediction. The procedure follows the same setup as Hesham *et al*. [7]

CNN

Our proposed model uses a CNN to extract useful features from the input image. The architecture is inspired by the architecture used in NVIDIA's DAVE-2 system [2]. The modified network takes a 160x350x3 image as input, followed by a cropping layer and a normalization layer. The cropping layer removes the top 50 pixels from the image, while the normalization layer scales the pixel values between -0.5 and 0.5. Next follows six convolutional layers, all using a ReLU activation function. The first three convolutional layers use a 5x5 filter, while the last three use a 3x3 filter. The first four convolutional layers use a stride of 2, while the last two use a stride of 1. The output of the last convolutional layer is flattened, resulting in a one-dimensional feature vector with 1024 elements.

LSTM

The output from the CNN is concatenated with the current navigational input and external state information, producing a vector of length 1033. The concatenated vector is connected to an LSTM layer with ten hidden states, that uses a sequence of feature extractions over time to produce a control signal. For each time step in the sequence, the LSTM layer sends its output to itself. Finally, at the last time step, the output is forwarded to a dense layer.

Steering Angle Classification

Following the same setup as Hesham *et al.* [7], the *Steer Predictor*-module used a classification layer containing ten neurons. Furthermore, a *tanh* activation is applied to the classification layer allowing the neurons to shape a sine wave with an amplitude of 1. The original steering angle corresponds to the phase shift, ϕ, of the sine wave. During training, the ground truth steering angle is encoded as a sine wave using Equation 2. Y_i is the encoded target value for output neuron i, ϕ is the raw steering angle, and ϕ_{max} is the maximum possible raw steering angle. The loss of the prediction is calculated as the RMSE between the predicted waveform and the encoded ground truth waveform. During deployment, the classification layer's output is decoded back to a steering angle. The decoding is done by fitting the classification layer's output to a sine function and returning its phase shift.

7 Experimental Setup

Real-time tests in CARLA We perform real-time tests by letting a model control a simulated vehicle in completely unseen environments. That is, the model is continuously fed with images from the vehicle's vantage point, while the model's predictions are applied as control signals for the vehicle. The model is to drive five different predefined routes, in five diverse weather conditions comprised of *Clear Noon*, *Clear Sunset*, *Heavy Rain Noon*, *Soft Rain Sunset*, and *Wet Sunset*. Route 1-3, positioned in *Town 2*, tests the model's ability to operate in urban environments containing traffic, intersections, traffic lights and speed limits. Route 4 and 5, positioned in *Town 4*, tests the model's ability to operate on highways and high-speed arias.

Experiment 1: Sequence Length The goal of experiment 1 was to find out how the number of hidden states in the LSTM cells affects the system's performance. Five models

were trained using 1, 5, 10, and 20 hidden states in the LSTM cells. Each trained model was evaluated by their performance from a single run of the real-time tests in CARLA.

Experiment 2: Size of Dataset The goal of experiment 2 was to clarify how the size of the training dataset affects the system's performance. Four models were trained using datasets of varying size. The different datasets all originated from the final, augmented and balanced dataset, but with an increasingly larger portion of the dataset randomly dropped. The models were trained on 20 %, 40 %, 60 %, and 80 % of the original training set. Each trained model was evaluated by their average route completion on a single run of the real-time tests in CARLA.

Experiment 3: Prediction Frequency The goal of experiment 3 was to determine how the model's prediction frequency affects the performance in the real-time tests. The prediction frequency limits how many control-signals the model can produce each second, thus restricting how fast the model is able to react to changes in the perceived environment. The model was exposed to the real-time tests in CARLA five times. For each run, the model's allowed prediction frequency was set to 30 Hz, 15 Hz, 10 Hz, 5 Hz, and 1 Hz, respectively. The models were evaluated by their average route completion.

Experiment 4: Extensive Real-time Tests The goal of experiment 4 was to evaluate the performance of the best model from *Experiment 1* extensively. The model was evaluated by its average performance on ten runs of the real-time tests in CARLA.

8 Experimental Results

Experiment 1: Sequence Length Table 2 shows the model's average route completions for four different sequence lengths in the model's recurrent module. Using a sequence length of 1, 5, 10, and 20, lead to an average route completion of 79.9%, 89.7%, 92.5%, and 76.8%, respectively.

Experiment 2: Size of Dataset Table 3 lists the average route completions for a model using four different training dataset sizes. The model using the original dataset had a route completion of 97.1%. Keeping 80% of the dataset lead to 74.1% average route completion, while further dropping the dataset by 20% lead to a 59.1% average route completion. After that, the performance decrease was less noticeable. Keeping 40% and 20% of the dataset lead to 60.0% and 55.1% average route completions, respectively.

Experiment 3: Prediction Frequency Table 4 shows the average route completions for the model tested with different prediction frequencies. The models using a 30 Hz or 15 Hz prediction frequency yielded the best results with an average route completion of 97.1%. The model with a 10 Hz prediction frequency had a slightly lower route completion of 95.1%. Using a prediction frequency of 5 Hz, 3 Hz, and 1 Hz, resulted in an average route completion of 86.1%, 76.2%, and 27.8%, respectively.

Seq. Length	Avg. Route Compl.
1	79.9%
5	89.7%
10	**92.5%**
20	76.8%

Table 2: The average route completions after using different sequence lengths in the model's recurrent module.

Dataset Size	Avg. Route Compl.
100%	**97.1%**
80%	74.1%
60%	59.1%
40%	60.0%
20%	55.1%

Table 3: The average route after dropping increasingly larger portions of the dataset.

Pred. Freq. [Hz]	Avg. Route Compl.
30	**97.1%**
15	**97.1%**
10	95.1%
5	86.1%
3	76.2%
1	27.8%

Table 4: The average route completions after reducing the model's allowed prediction frequency.

Town	Clear Noon	Clear Sunset	Heavy Rain Noon	Soft Rain Noon	Clear Wet Sunset	Avg.
Town 2	100%	96.5%	80.1%	100%	100%	95.3%
Town 4	100%	100%	83.8%	94.9%	41.3%	84.0%

Table 5: The average route completions in each weather condition from Experiment 4.

Experiment 4: Extensive Real-time Tests Table 5 shows the average route completions in each weather condition over ten real-time tests. The model had an average route completion of 95.3% and 84.0% in Town 2 and Town 4, respectively. Table 6 lists the number of failures per traveled km in Town 2 and Town 4. We divided the failures into three categories: minor, moderate, and severe. Lane touches and sidewalk touches were minor failures, while object collisions, navigational input violation, and rear-end vehicle collisions were considered moderate failures. Severe failures were comprised of front-end vehicle collisions, entering the oncoming lane, and traffic light violations.

In Town 2, the model had the highest number of failures in all categories in *Heavy Rain Noon*, while in Town *Clear Wet Sunset* was the weather condition with the highest frequency of failures.

Parts of experiment 4 were recorded to create a video[2] previewing the system's performance.

	Failures					
Weather Condition	Town 2			Town 4		
	Minor	Moderate	Severe	Minor	Moderate	Severe
Clear Noon	0	0.2	0.2	0.68	0.15	0
Clear Sunset	0.41	0	0.21	0.61	0.68	0
Heavy Rain Noon	**3.49**	**3.99**	**2.00**	0.63	0.54	0
Soft Rain Noon	1.2	0.8	0	0.24	0.08	0.08
Clear Wet Sunset	1.8	1.6	0	**1.47**	**1.28**	**0.73**
Total	1.30	1.21	0.42	0.63	0.44	0.09

Table 6: Experiment 4 – Number of failures per traveled km in Town 2 and Town 4.

[2]A preview of the overall performance can be seen here: `www.tinyurl.com/e2e-example`

9 Discussion

Experiment 1: Sequence Length

The results from *Experiement 1* showed that a small sequence length could lead to faster reactions. However, the network could only learn short term temporal dependencies and was consequently more susceptible for individually misclassified frames. A longer sequence length, on the other hand, led to smoother behavior, and thus more stable steering predictions. Still, it could also increase the probability of learning wrong long-term temporal dependencies. Overall, the system performed most reliably when using a sequence length of 10.

Experiment 2: Size of Dataset

The results from the second experiment showed that a small reduction of dataset size impacted the performance negatively. Reducing the dataset by 20% and 40% resulted in a performance decrease of 24% and 39% respectively. However, reducing the dataset further did not result in any drastic performance decrease. The average route completion stayed approximately the same after reducing the dataset by 40%, 60%, and 80%.

The results also suggest that the system were more sensitive to changes in dataset size in complex weather conditions. The average route completion in *Clear Noon* and *Clear Sunset* were only decreased by 22.5% after reducing the dataset by 80%. In *Sof Rain Noon* and *Clear Wet Sunset*, the same reduction resulted in a decrease in average route completion of 43.5%. In *Heavy Rain*, the average route completion was decreased by 92.3%.

Experiment 3: Prediction Frequency

The experimental results revealed that the reduction of prediction frequency from 30 Hz to 15 Hz did not result in any performance decrease. Both models were able to achieve an average of 97.1% route completion. When further reducing the prediction frequency to 10 Hz, a small performance reduction of 2% was observed. However, reducing the frequency to less than 10 Hz resulted in a significant performance decreases. Using a prediction frequency of 5 Hz, 3 Hz, and 1 Hz respectively, resulted in an 11%, 20.9%, and 69.3% worse performance than the best model. Furthermore, the results showed that the system is more sensitive to reduction of prediction frequencies in complex weather conditions.

Experiment 4: Extensive Real-time Test

Town 2 - Urban Environment Regardless of weather conditions, the system always stopped at red lights, never ignored a navigational input, and followed the current speed limit. Moreover, it performed stable lane following and always tried to position itself in the center of the lane. When drifting out of the lane, the system corrected its trajectory both quickly and smoothly, never experiencing any oscillating steering predictions. In situations where the system had to decrease its speed, it tried to let go of the throttle first. If that was insufficient, the system applied the brakes as well.

The results indicate that *Hard Rain Noon* was the most challenging weather condition. The system displayed a smooth driving behavior on straight stretches, but the steering was uneven in turns. The system had trouble with detecting cars in front of it, mainly black and dark grey cars. The light conditions were poor in *Hard Rain Noon*, which made the dark-colored cars blend more in with the environment.

Town 4 - Highway Regardless of weather conditions, the system always stopped at red lights and followed the current speed limit. The system adjusted its distance to the car in front of it by adjusting the throttle and was able to handle traffic congestions in both low-speed and high-speed areas.

Clear Wet Sunset turned out to be the most challenging weather condition in *Town 4*, and the system had its lowest average route completion in this weather condition. The sunlight reflected off the wet parts in the road, making it especially challenging to detect the lane lines. As a result, the system had more trouble with staying in the lane and drove out of the road four times throughout the experiment. Additionally, the system experienced four object collisions, two rear-endings, and one route deviation.

10 Conclusion

In this paper, we explore an end-to-end system's ability to drive autonomously in diverse, simulated environments. We propose a system architecture consisting of two modules: a *Steer Predictor* and a *Throttle-Brake Predictor*. Each module combines a traditional CNN, inspired by NVIDIA's DAVE-2 system [2], with an LSTM layer to facilitate the learning of both spatial and temporal relationships. The system is trained using expert driving data from various simulated settings and we evaluate our proposed model by presenting extensive real-time experiments in unseen simulated environments.

Our proposed model is able to operate successfully in urban environments containing other vehicles, speed limits, and traffic-light regulated intersections. The system always stopped at red lights, always tried to execute the navigational input, and followed the current speed limit. On highways, the real-time tests demonstrated that the system was able to perform stable lane following in most weather conditions and that it was capable of handling traffic congestions in both low-speed and high-speed areas.

The system is able to operate in a range of different weather conditions. It excelled in dry weather during the daytime, making few mistakes. In more challenging weather conditions, like soft rain, the system was more susceptible to failures, which could lead to the occasional rear-end collision. In even harsher weather conditions, such as heavy rain, the system was more likely to perform severe failures, such as entering the oncoming lane or front-end collisions. Nonetheless, the system was able to navigate remarkably well in harsh weather conditions where even humans could struggle.

In this paper, we also aimed to explore some of the uncertainties regarding the implementation of end-to-end systems. The experimental results showed that the system was sensitive to small reductions in dataset size. However, the system was able to operate somewhat successfully even with a severely reduced dataset. Furthermore, the results suggest that the system performed to its full potential when using a prediction frequency of 15 Hz or higher. Finally, it was found that using ten hidden states for the system's recurrent module was optimal for the overall performance.

Even though the system experienced several severe failures during testing, our real-time tests reflects great potential in using end-to-end systems to achieve fully autonomous vehicles.

References

[1] Hege Haavaldsen, Max Aasboe, and Frank Lindseth. Autonomous vehicle control: End-to-end learning in simulated urban environments, 2019.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[3] Dean A. Pomerleau. Advances in neural information processing systems 1. chapter ALVINN: An Autonomous Land Vehicle in a Neural Network, pages 305–313. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[4] Y. Lecun, E. Cosatto, J. Ben, U. Muller, and B. Flepp. Dave: Autonomous off-road vehicle control using end-to-end learning. Technical Report DARPA-IPTO Final Report, Courant Institute/CBLL, http://www.cs.nyu.edu/˜yann/research/dave/index.html, 2004.

[5] Felipe Codevilla, Matthias Müller, Alexey Dosovitskiy, Antonio López, and Vladlen Koltun. End-to-end driving via conditional imitation learning. *CoRR*, abs/1710.02410, 2017.

[6] C. Hubschneider, A. Bauer, M. Weber, and J. M. Zöllner. Adding navigation to the equation: Turning decisions for end-to-end vehicle control. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8, Oct 2017.

[7] Hesham M. Eraqi, Mohamed N. Moustafa, and Jens Honer. End-to-end deep learning for steering autonomous vehicles considering temporal dependencies. *CoRR*, abs/1710.03804, 2017.

[8] Eder Santana and George Hotz. Learning a driving simulator. *CoRR*, abs/1608.01230, 2016.

[9] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio López, and Vladlen Koltun. CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938, 2017.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.